7-2004

Port yard storage optimization

Ping CHEN
University of Maryland at College Park

Zhaohui FU

Andrew LIM

Brian RODRIGUES
Singapore Management University, brianr@smu.edu.sg

Published in IEEE Transactions on Automation Science and Engineering, 2004 July, Volume 1, Issue 1, Pages 26-37
https://doi.org/10.1109/tase.2004.829412

Port Yard Storage Optimization

Ping Chen, Zhaohui Fu, *Student Member, IEEE*, Andrew Lim, *Senior Member, IEEE*, and Brian Rodrigues

***Abstract*—The port yard storage optimization problem (PYSOP) originates from space allocation needs at the Port of Singapore. Space allocated to cargo is to be minimized in a designated yard within a time interval. The problem is akin to a packing problem in space and time, but where shapes packed and constraints are particular to port operations. Further, space requests can change within the time interval in which it is requested. This basic problem is generic to port operations and may find applications elsewhere. The PYSOP is NP-hard, but we propose a number of metaheuristics. Extensive experiments were conducted and good results obtained.**

***Note to Practitioners*—The Port of Singapore is one of the busisest ports in the world where competing pressures for land use and competition from other regional and international ports force port planners to make best use of available land. Factors that impact storage capacity include stacking heights, net storage area available, storage density (containers per acre), dwell times for empty containers and breakbulk cargo. In studying its operations to find better ways to utilize storage space within the dynamic environment of the port, we narrowed storage problems down and focused on the central allocation process in storage-operations improvement which would allow for better utilization of space. In this process, requests are made from an operations unit which coordinates ship berthing and ship-to-apron loading as well as apron-to-yard transportation. Each request is for a set of spaces within a yard required in a single time interval. If any space is allocated to the request, this space cannot be freed (released) until the request is completed, that is, until the end time point of the time interval. The problem is akin to a packing problem in space and time, but where shapes packed and constraints are particular to port operations. Further, space requests can change within the time interval in which it is requested. This basic problem is generic to port operations and may find applications elsewhere. The PYSOP is NP-hard for which we propose a number of metaheuristics. Extensive experiments were conducted and good results obtained.**

***Index Terms*—Automation, packing, port logistics, scheduling.**

I. Introduction

STORAGE is an important constraining factor in logistics management for many ports. Factors that impact terminal storage capacity include stacking heights, net storage area available, storage density (containers per acre), dwell times for empty containers, and breakbulk cargo. The Port of Singapore, which is one of the world's largest in terms of shipping tonnage handled, faces space constraints in a unique way as it is located on a small island. The competing pressures for land use and competition from other regional and international ports forces port planners to make best use of available land. As such, the optimization of storage of cargo in its available yards is crucial to its operations and commercial viability. In studying its operations to find better ways to utilize storage space within the dynamic environment of the port, we narrowed storage problems down and focused on the central allocation process in storage-operations improvement, which would allow for better utilization of space. In this process, requests are made from an operations unit which coordinates ship berthing and ship-to-apron loading as well as apron-to-yard transportation. Each request is for a set of spaces within a yard required in a single time interval. If any space is allocated to the request, this space cannot be freed (released) until the request is completed, that is, until the end time point of the time interval. A variant of this situation is when space requirements are allowed change during the time interval of any given request. This may arise from changes in the other components of the storage process and is part of the dynamic backdrop of port operations. The final objective is to pack all such requests into a yard space of minimum area within the given constraints.

We propose a model for basic port storage optimization which represents a generic problem we expect will be useful to other port managements. Sabria and Daganzo [1] have given a bibliography on port operations studies with the focus on berthing and cargo-handling systems. On the other hand, traffic and vehicle-flow scheduling on landside up to yard points have been studied well (see, for example, Bish *et al.* [2]). Other than studies such as Gambardella *et al.* [3], which address spatial allocation of containers on a terminal yard using simulation techniques, there has been little direct study on yard space allocation as described in this paper. Further, although related to packing optimization—in particular two-dimensional (2-D) packing—the port yard storage optimization problem (PYSOP) is different from these. It has, for example, a nondecreasing space request constraint and the degree of freedom allowed for objects to be moved into positions differs from packing routines.

We describe the basic model in Section II and provide a transformation of it into a graph problem in Section III. Metaheuristics are used to obtain solutions for the problem: tabu search (TS) in Section V; simulated annealing (SA) in Section VI; and squeaky wheel optimization (SWO) in Section VII. In Section VIII, genetic algorithms (GAs) with various crossover operators are introduced, and in Section IX we provide results of experiments. We conclude this work in Section X.

Manuscript received April 23, 2003.

P. Chen is with the Department of Computer Science, University of Maryland, College Park, MD 20742 USA (e-mail: pchen@cs.umd.edu).

Z. Fu is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: zfu@princeton.edu).

A. Lim is with the Department of Industrial Engineering and Engineering Mgmt., Hong Kong University of Science and Technology, Hong Kong (e-mail: alim@ust.hk, iealim@ust.hk).

B. Rodrigues is with the School of Business, Singapore Management University, Singapore 259759 (e-mail: br@smu.edu.sg).

Digital Object Identifier 10.1109/TASE.2004.829412

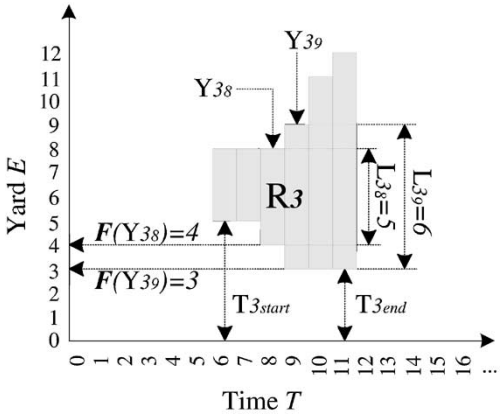


Fig. 1. *Valid* request R_3.

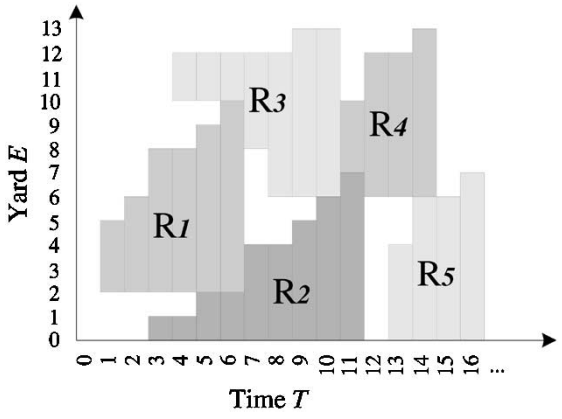


Fig. 2. Five *valid* requests.

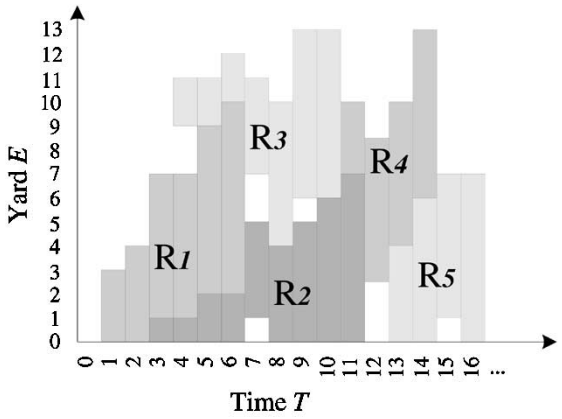


Fig. 3. Five *invalid* requests.

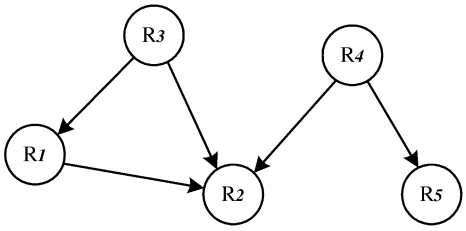


Fig. 4. Graph transformation of Fig. 2.

II. Problem Definition

The objective in the PYSOP is to minimize the utilized yard space while satisfying space requirements. The problem can be described as follows.

Let R be a set of n yard space requests (as described above), and let E represent the yard or a section of it. Each request $R_i \in R, i = 1, \ldots, n$, comprises a series of space requirements, Y_{i_j}, each of length L_{i_j}, for $j \in [T_{i_{\text{start}}}, T_{i_{\text{end}}}]$, where the latter time interval is defined by the request R_i.

A mapping F is chosen such that, for each i, j, $F(Y_{i_j}) = e_k$, for some position $e_k \in E$, which satisfies the constraints: For all $p, q \in [T_{i_{\text{start}}}, T_{i_{\text{end}}}]$ such that $p \leq q, F(Y_{i_p}) \geq F(Y_{i_q})$, and $F(Y_{i_p}) + L_{i_p} \leq F(Y_{i_q}) + L_{i_q}$. These constraints provide for the fact that space requests are nondecreasing in time since we will assume that requests for space over the time window considered are for loading of cargo into the yard or a section of it. This is the case in this study where the yard or section of it is designated to hold all cargo bound for a certain ship(s). Within this time, no cargo is taken out of the designated area. The function F maps requests to positions in the yard and acts as a selection function. The objective is then to minimize

$$\max_{\substack{Y_{i_j} \in R_i \\ i=1,\ldots,n}} (F(Y_{i_j}) + L_{i_j})$$

over all possible mappings F. With this objective and constraints, all requests received are accommodated within a minimum amount of yard space.

Example: We provide a simple example to illustrate the problem. Fig. 1 shows a layout with only one *valid* request, say R_3. The yard E is treated as an infinite straight line. Time T is taken to be a discrete variable with a minimum unit of 1 and R_3 has six space requirements within the time interval $[T_{3_{\text{start}}} = 6, T_{3_{\text{end}}} = 11]$. The final position for Y_{3_8} and Y_{3_9} are $F(Y_{3_8}) = 4$ and $F(Y_{3_9}) = 3$, respectively. The corresponding output for R_3 is then (5, 5, 4, 3, 3, 3), where we note that both constraints are satisfied since $F(Y_{3_8}) \geq F(Y_{3_9})$ and $F(Y_{3_8}) + L_{3_8} \leq F(Y_{3_9}) + L_{3_9}$. Here, the objective maximum is achieved at $Y_{3_{11}}$, with a value of $F(Y_{3_{11}}) + L_{3_{11}} = 13$.

We refer to such requests as "stair-like shape" (SLS) requests throughout this study. Fig. 2 shows five such valid requests where the minimum yard space required is 13. Although the packing in Fig. 3 appears compact, all allocations shown are *invalid* since the nondecreasing requests constraint prescribed for the problem is violated.

Theorem 1: The PYSOP is NP-hard.

The ship berthing problem (SBP) given in [4] is similar to the PYSOP except that all requests are of rectangular shape rather than the more general SLS here. Reference [5] provided a proof that the SBP is NP-hard (in the strong sense) by reducing the set partitioning problem to the SBP. Since the SBP is a special case of PYSOP when we do not consider end-berth and inter-ship clearances, the PYSOP is also NP-hard (in the strong sense). In fact, in [4], both the end-berth and inter-ship clearance are taken as part of the ship in the proof that the SBP is NP-hard. ■

III. Graph Transformation

Fig. 2 illustrates a problem geometrically. In our solution approach, we first transform this layout into a graph which results in Fig. 4. Each request R_i is represented by a vertex and there

exists an edge E_{ij} connecting R_i and R_j if R_i and R_j have an overlap in time. The direction of the edge determines the relative position of the two requests in the yard itself. Using Fig. 2 again as an example: both R_1 and R_2 require space at time 3–6 and hence, in Fig. 4, there is an edge connecting R_1 with R_2. Since R_1 is located above R_2, the direction of the edge is from R_1 *toward* R_2. We call this directed edge E_{12}. Clearly, the transformed graph is a directed acyclic graph (DAG) where each vertex R_i can be assigned an acyclic label (AL) $\mathrm{AL}(R_i)$, and where, for each directed edge $E_{jk}, \mathrm{AL}(R_j) < \mathrm{AL}(R_k)$. We note here that each AL $\mathrm{AL}(R_i), i = 1, \ldots, n$ has a unique value.

Lemma: For each feasible layout in a yard, there exists at least one corresponding AL assignment of the vertices in the graph representation.

Proof: A simple constructive proof is provided. Given any feasible layout L.

ASSIGN-AYCLIC-LABELS (L)
1 $C := 0$
2 **while** L is not empty
3 pick one "free" SLS F
4 remove F from L
5 $\mathrm{AL}(F) := C$
6 $C := C + 1$

A "free" SLS is the one with no other SLS above it so that there is no obstacle blocking it from being popped out through the top of the layout. Again, consider Fig. 2 as example. In the first iteration of the loop R_3 and R_4 are the only two "free" SLSs. If we assign $\mathrm{AL}(R_3) = 0$, in the second iteration R_1 will become another "free" SLS. The process continues until there are no more SLSs left in L. ■

We note that an AL assignment is a partial ordering and each physical layout can correspond to more than one AL assignment. For example, $[R_1 : 2, R_2 : 3, R_3 : 0, R_4 : 1, R_5 : 4]$ and $[R_1 : 2, R_2 : 4, R_3 : 1, R_4 : 0, R_5 : 3]$ are two possible AL assignments of the same layout.

The presence of alternative optima can help in heuristic search since it increases the possibility of finding a good solution. However, this one-to-many relationship between physical layout and AL assignments in the graph representation can cause difficulty in heuristic search. Two different AL assignments corresponding to the same physical layout are not alternative optima since they are in fact identical and correspond to the same optimum. Our heuristic methods tend to identify good patterns which can potentially lead to better solutions while exploring the search space. Different looking solutions, which correspond to the same physical layout, will make it difficult for such a heuristic to locate the correct patterns. For example, if we perform a crossover operation—in a GA approach discussed later—on two different AL assignments that correspond to the same physical layout, the correct patterns can become distorted. Here, by removing such ambiguities in solution representation, we do not eliminate alternative optima.

Such difficulties are avoided by normalizing AL assignments. When there is more than one SLS to be popped out, we break the tie by selecting the SLS with the smallest label. A topological sort with this tie breaker will provide each layout with a unique AL assignment. We take it here that our solutions are represented by their normalized and unique AL assignments.

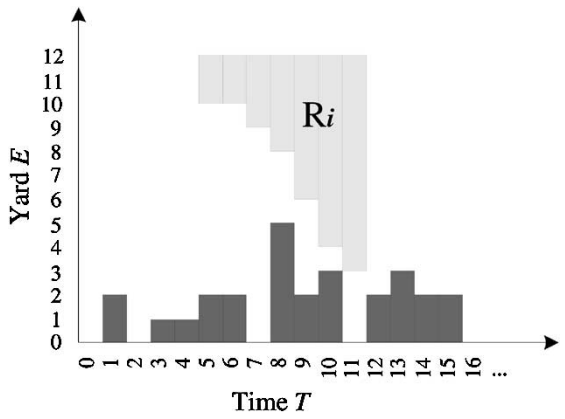


Fig. 5. Before dropping: R_i is ceiling aligned.

Since each physical layout now has a unique AL assignment, the optimal layout will have an optimal AL assignment and our goal is to find such an optimal AL assignment. A major operation—the evaluation of a given AL assignment—turns out to be a difficult process. In SBP [4], [6], [7], a longest path algorithm on a DAG was employed to find a minimum berth length. However, the PYSOP deals with SLSs whose relative position and distance cannot be calculated in a straight-forward way, unlike the case for rectangles. We resort to the use of a recursive procedure to find the minimum yard needed for a given AL assignment (A), which is given in the following:

1) **EVALUATE-SOLUTION** (A)
2) **while** there exists unallocated SLSs
3) pick SLS S with largest AL
4) Drop $(S, S_{\mathrm{end}}, 0)\{S_{\mathrm{end}}\mathrm{is\ the\ last\ time\ slot\ used\ by}\}$
5) **for each** time T_i
6) **if** $T_i > L$
7) $L := T_i$
8) **return** L

Drop $(S, t, l)\{l\,\mathrm{is\ the\ lowest\ position}\ S\ \mathrm{can\ drop\ to}\}$
1) $L :=$ lowest position to drop all stairs(time t')
2) **if** $L < l$
3) $L = l$
4) **for all** stair s after $t' - 1$
5) drop s to position L
6) Drop $(S, t' - 1, L)$

The recursive DROP function above uses a greedy technique to drop a given SLS to a position which is as low as possible. Figs. 5–7 illustrate this: R_i has seven space requirements starting from time 5 until time 11 and is first initialized to be aligned to the ceiling before the process starts (Fig. 5). From time 5 to time 11, we find the maximum distance that each "stair" can be dropped, without exceeding a lower bound of 0 where the minimum among all the maximum possible drops is used. In this case, the minimum distance of 1 is given at time 10 and hence every stair is shifted down by 1 (Fig. 6). Because of the initial ceiling alignment, no further shifting downwards

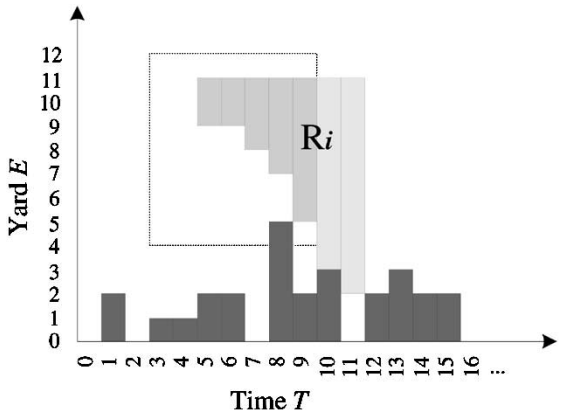


Fig. 6. Each stair of R_i drops by 1. Stairs at time 10 and 11 are in their final positions. Those stairs which can drop further are in a dark color surrounded by a rectangle.

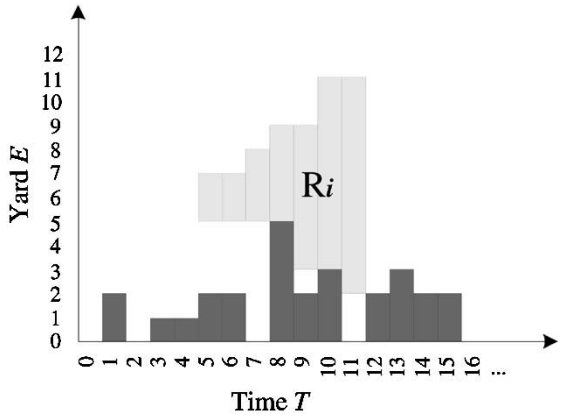


Fig. 7. Final layout.

is required for all stairs at time 10 (inclusive) onwards. Note that the surface was touched at time 10.

Stairs from times 5 to 9, which are surrounded by a rectangle in Fig. 6, can be dropped further and will be recursively dropped but now with a lower bound of 3, which is the height of the previously touched surface. The dropping process is completed after recursions at times 8, 7, and 6 and the final layout is shown in Fig. 7. Note the worst case time complexity for DROP is $n \times t$, where n is the number of requests and t is the average time span for all requests.

Lemma 2: For a given AL assignment, the greedy drop procedure always gives a layout which uses minimum yard space.

Proof: Compare the solution G obtained by the greedy drop approach with any arbitrary optimal solution O and the following algorithm:

1) **COMPACT** $(A, G, O))$
2) **let** $L :=$ set of SLSs;
3) **while** L is not empty
4) pick SLS S with largest AL
5) **for** $(i = S_{\text{begin}}; i \leq S_{\text{end}}; i++)$ {Loop through the time slots used by S}
6) **let** $G_{s_i} :=$ position of S_i in G
7) **let** $O_{s_i} :=$ position of S_i in O
8) **if** $O_{s_i} > G_{s_i}$
9) $O_{s_i} := G_{s_i}$

The algorithm COMPACT transforms any optimal solution into a solution that can be obtained by the greedy approach without an increase in the amount of the yard used. Here, line 7 in COMPACT depends on the fact that no other solution allocates S_i to a lower position than greedy approach does. ■

We have developed a one-to-one relationship between any physical layout and the corresponding AL assignment. The problem that remains now is one of finding the optimal AL assignment.

IV. Metaheuristics

Over the last 20 years, metaheuristics have been widely and increasingly used to solve complex combinatorial optimization problems. The PYSOP is related to packing optimization, and in particular, to 2-D packing where we have seen the application of many metaheuristic approaches. Hopper and Turton [8] have given a comprehensive review of metaheuristics applied to 2-D packing problems and extensive references of works that use metaheuristics for such packing problems. As we have mentioned, the PYSOP is different from these problems, but, as with these, the complexity of the PYSOP suggest the use of metaheuristics which we employ in the following sections.

V. Tabu Search

One of the more successful metaheuristics developed and used is TS first proposed by Glover in [9]. TS is a local search heuristic that uses the best neighborhood move that is not "tabu" active to move out from local optimum by incorporating *adaptive memory* and *responsive exploration* [10]. By the different usage of memory, TS has been classified into two categories: TS with short-term memory (TSSTM) and TS with long-term memory (TSLTM) [9], [11].

TSSTM is the simpler method, in which memory consists of a tabu list. Such an adaptation is also known as a *recency*-based TS. TSLTM, however, uses more advanced TS techniques including intensification and diversification strategies [12]. The TS implementation we use always terminates after a prescribed number of nonimproving iterations.

A. Tabu Search With Short-Term Memory

The TSSTM implementation given here consists of two main components: neighborhood search and the use of a tabu list. A neighborhood solution can be obtained by swapping positions within AL assignments. For example, [2, 3, 0, 1, 4] is a neighborhood solution of [1, 3, 0, 2, 4] by swapping the positions of 1 and 2. However, after normalization, some swaps may be identical with the original AL assignment and are excluded from the neighborhood for the sake of efficiency.

Solutions represented by AL assignments are sequences of numbers where the cardinality of these sequences is the number of requests made. Selected attributes that occur in solutions recently visited are labeled *tabu active* and kept in tabu lists, and solutions that contain tabu-active elements become tabu.

B. Tabu Search With Long-Term Memory

We implemented TSLTM in two phases: the diversification and intensification phases. We used two diversification tech-

niques: one randomly restarts and the other randomly picks a subsequence and inserts it into a random position. For example, [0, 1, 2, 3, 4] can be transformed into [0, 3, 2, 1, 4] if [2, 3] is chosen as the subsequence and its reverse (or its original, if random) is inserted back into the position in preceding 1. Intensification is similar to that for the TSSTM. Moreover, TSLTM uses a *frequency*-based memory by recording both *residence* frequency and *transition* frequency of visited solutions. In implementation, residence frequency is taken to be the number of times $\mathrm{AL}(R_i)$ is strictly less than $\mathrm{AL}(R_j)$ in the selected solution, for any $i, j \in \{1, \ldots, n\}$. The transition frequency is taken as the sum of the improvements when $\mathrm{AL}(R_i)$ is swapped with $\mathrm{AL}(R_j)$, where this sum can be positive or negative.

Diversification and intensification are interleaved, and during either phase, the residence frequency and transition frequency is updated according to the current selected solution. The objective function has three contributors, besides the yard space length required, both residence frequency and transition frequency are used to evaluate solutions. Higher residence frequency indicates that an attribute is more desirable. In contrast, a high transition frequency can indicate that the associated attribute changes in the solution because of its fine tuning function [9]. In this context, the transition frequency can be interpreted as a measure of volatility. The yard length required has the major impact on our objective function. Residence frequency is the second most important contributor while transition frequency was not impactful as found in our experiments.

VI. Simulated Annealing

SA [13]–[15] is a general optimization method which stochastically simulates the cooling of a physical solid providing the algorithmic means for exploiting the connection between thermodynamic behavior and the search for global optima. A key feature is that it provides means to escape local optima by allowing hill-climbing moves. We used the following SA algorithm on the PYSOP.

Step 1) Choose an initial temperature T_0 and a random initial starting configuration θ_0. Set $T = T_0$ and define the objective function (energy function) to be $En()$ and the cooling schedule to be σ.

Step 2) Propose a new configuration θ' of the parameter space, within a neighborhood of the current state θ, by setting $\theta' = \theta + \phi$ for some random vector ϕ.

Step 3) Let $\delta = En(\theta') - En(\theta)$. Accept the move to θ' with probability

$$\alpha(\theta, \theta') = \begin{cases} 1, & \text{if } \delta < 0 \\ \exp\left(-\frac{\delta}{T}\right), & \text{otherwise.} \end{cases}$$

Step 4) Repeat Step 2 and 3 for K of iterations, until it is deemed to have reached the equilibrium.

Step 5) Lower the temperature by letting $T = T \times \sigma$ and repeat Steps 2–4 until certain stopping criterion, for our case $T < \epsilon$, is met.

Due to the logarithmic decrement of T, we set $T_0 = 1000$. The energy function is defined as the length of the yard required and the probability $\exp(-\delta/T)$ is the Boltzmann factor. The number of iterations K is proportional to the input size n and a neighborhood is defined as in the TS algorithm.

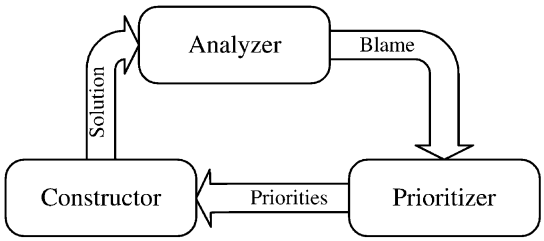


Fig. 8. C-A-P cycle in SWO.

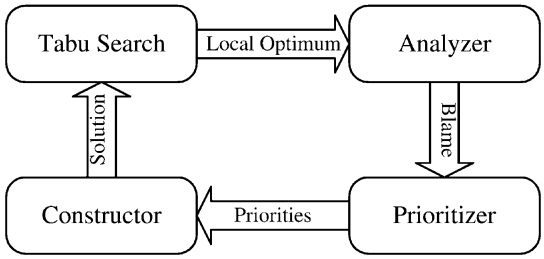


Fig. 9. Modified SWO with TS.

VII. "Squeaky Wheel" Optimization

SWO is a relatively newer heuristic. In SWO, introduced by Joslin and Clements [16] and [17], and (see also [18] and [19]), a construction algorithm first processes each element of a solution in an order that is determined by priorities assigned to each element based on certain criteria. The solution is then examined to determine which elements are positioned disadvantageously. These elements are deemed to "squeak" because they contribute negatively to the objective function of the solution. These "trouble" elements are then advanced to the front in the ordered priority list so that the construction algorithm handles them earlier when the next solution is constructed. This process of constructing, analyzing and reordering is repeated, producing a variety of candidate solutions to the problem at hand. In favorable situations, near optimal or even optimal solutions can be found with this procedure.

The basic approach in SWO is to form a construct-analyze-prioritize (C-A-P) three-component cycle. The "*constructor*" uses priorities assigned to construct a solution, employing a greedy algorithm. The "*analyzer*" assigns a numerical value as a "blame" factor to each element that has contributed to the shortcomings in the solution constructed in previous step. The "*prioritizer*" then modifies the priority list according the blame factor assigned for each element, by moving elements with greater blames to the front of the list.

In the SWO for the PYSOP, a greedy algorithm is used to construct a solution according to certain priorities (initially randomly generated) which is then analyzed to find the "trouble makers," i.e., those elements that, if improved, are likely to improve the objective function value. The results of the analysis are used to generate new priorities that determine the order in which the greedy algorithm constructs the next solution. This C-A-P cycle continues until a prescribed limit is reached or an acceptable solution is found.

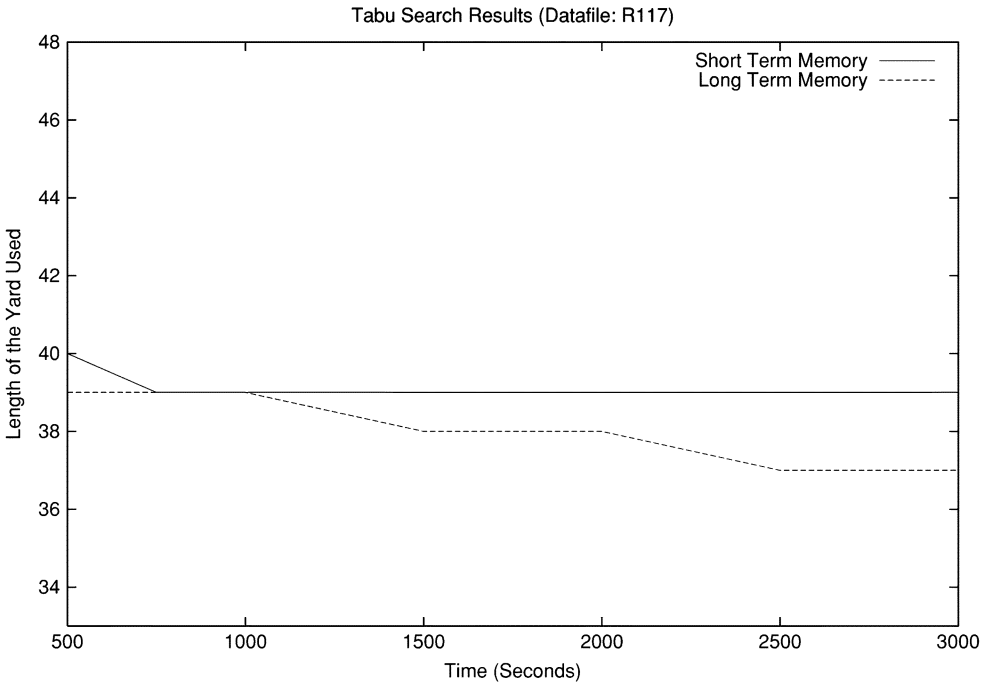


Fig. 10. Results obtained by TS. Data Set: R117.

From another perspective, SWO can be viewed as operating on two search spaces: the solutions and prioritizations spaces. Successive solutions are only indirectly related through reprioritization that result from analyzing prior solutions. Similarly, successive prioritizations are generated by constructing and analyzing solutions.

Fig. 8 provides an illustration of the components of SWO [16]. Information, including solutions and priorities are passed around in the C-A-P cycle. The SWO approach was implemented in our problem and gave good results. The system initiates with a random solution, which is a random AL assignment given by the *constructor*. The normalization routine is applied before the solution is passed to the *analyzer*, which will evaluate the solution by applying the dropping routine. If the best known result (yard length) is B, then, a threshold T is set to be $B - 1$. The blame factor for each request is the sum of the space requirements that exceed the threshold T. All the blame information is passed to the *prioritizer*, which is, in our case, a priority queue. When the control is handed over to the *constructor* again, it will delete the elements from the priority queue and immediately drop them in to the yard. In other words, the *constructor* works closely with the *prioritizer* to generate new solutions according to the priorities. SLSs with higher priority will be assigned smaller ALs. A tie is broken by considering ALs in the previous assignment. After all SLSs are assigned ALs, the latter are normalized. This tie breaker helps avoid cycling in the search process and experiments showed that the normalization routine plays a very important role in SWO.

The performance of the SWO can be further improved if we embed a "quick" TS technique, or a TSSTM into the SWO. We denote this modified SWO algorithm by: $\mathrm{SWO} + \mathrm{TS}$. A flowchart illustrating this is shown in Fig. 9. Here, the *constructor* will pass its solution to a TSSTM processor, which performs a quick local search and then passes the local optimum to the *analyzer*. Experiments show that this modification provides considerable improvement over the original SWO approach.

VIII. Genetic Algorithms

GAs are search procedures that use the mechanics of natural selection. The GA approach was first developed by Holland in the 1970s [20]. A GA is an iterative procedure that consists of a constant-size population of individuals, each one represented by a finite string of symbols, known as the *chromosome*, encoding a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at hand. The symbol alphabet used is often binary, though more and more other representations have been used recently [21].

The classical binary representation is not suitable in PYSOP which uses a list of ALs as solution representations. The solution space consists permutations of these numeric labels and a binary representation of these will not provide any advantage. In our approach, we adopted vector representations in which the AL assignment is used directly as the chromosome in the genetic evolution process. Two basic genetic operators used in our approach are the crossover and mutation operators:

A. Crossover Operators

Using an AL assignment as the chromosome, we implemented three crossover operators which are tailored to suit the problem domain in the PYSOP. These are: classical crossover with repair; partially-mapped crossover (PMX) and cycle crossover.

The classical crossover operator builds offspring by appending the head from one parent with the tail from the other parent, where the head and tail come from a random cut of the

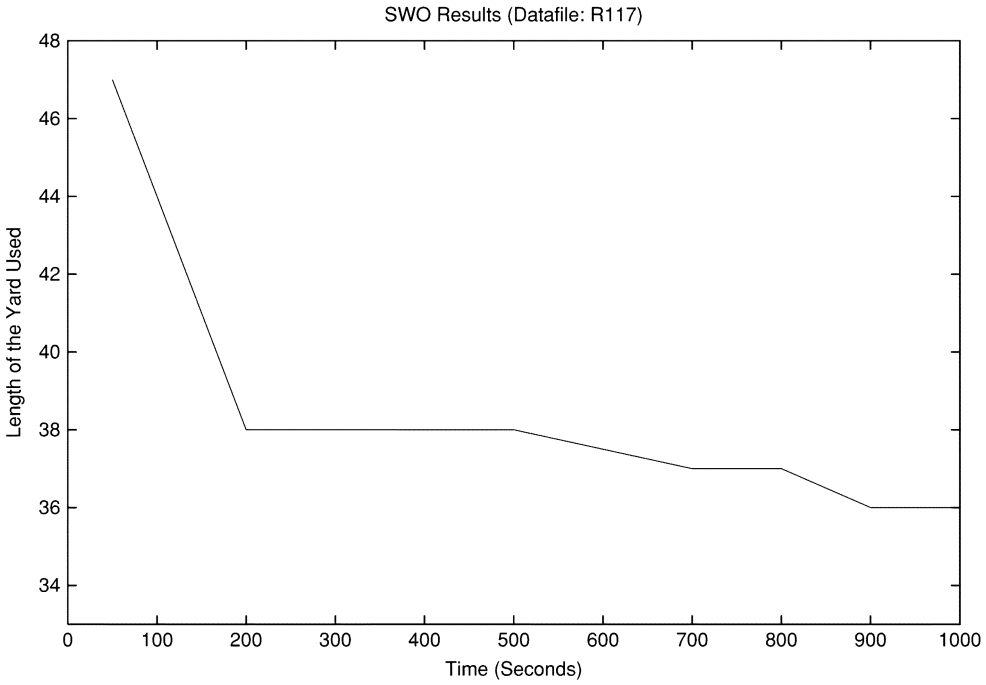


Fig. 11. Results obtained by SWO. Data Set: R117.

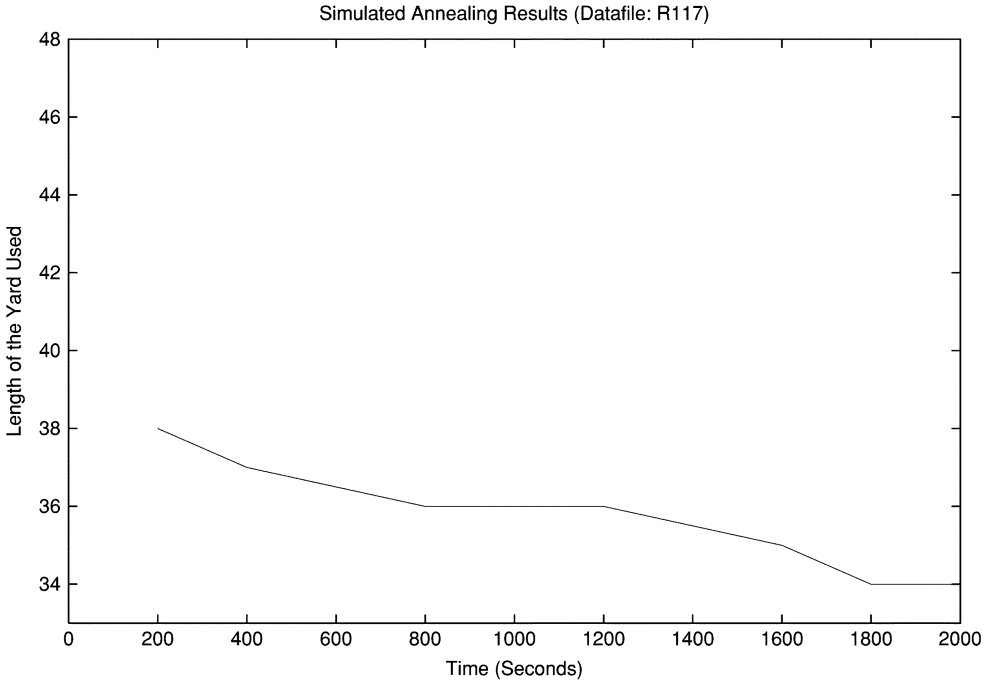


Fig. 12. Results obtained by SA. Data Set: R117.

parents' chromosomes. A repair procedure may be necessary after the crossover is effected [21].

PMX was first used in [22] to solve the traveling salesman problem (TSP). We made several adjustments to accommodate our chromosome (AL assignment) representation. The modified PMX builds an offspring by choosing a subsequence of an AL assignment from one parent and preserving the order and position of as many ALs as possible from the other parent.

Original cycle crossover (CX) was proposed in [23], again for the TSP problem. Our CX builds offspring in such a way that each AL (and its position) comes from one of the parents.

Our experiments show that classical crossover and CX have a stable but slow improvement rate while PMX demonstrates an oscillating but fast convergence pattern. In later experiments, the majority of crossovers were executed using PMX. Classical crossover and CX are applied, but with lower probabilities.

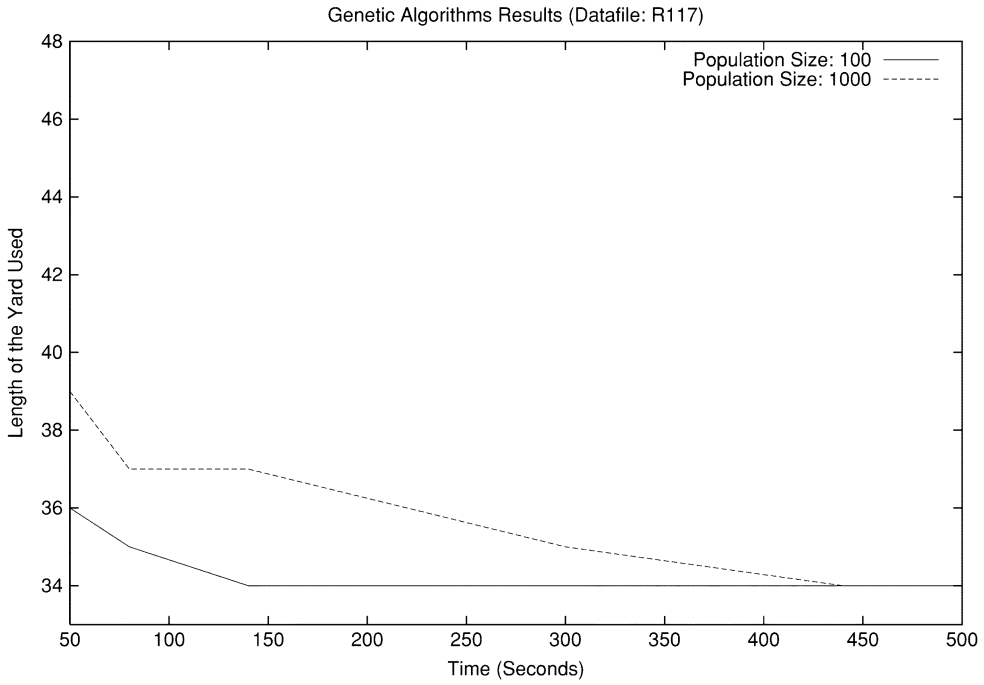


Fig. 13. Results obtained by the GAs. Data Set: R117.

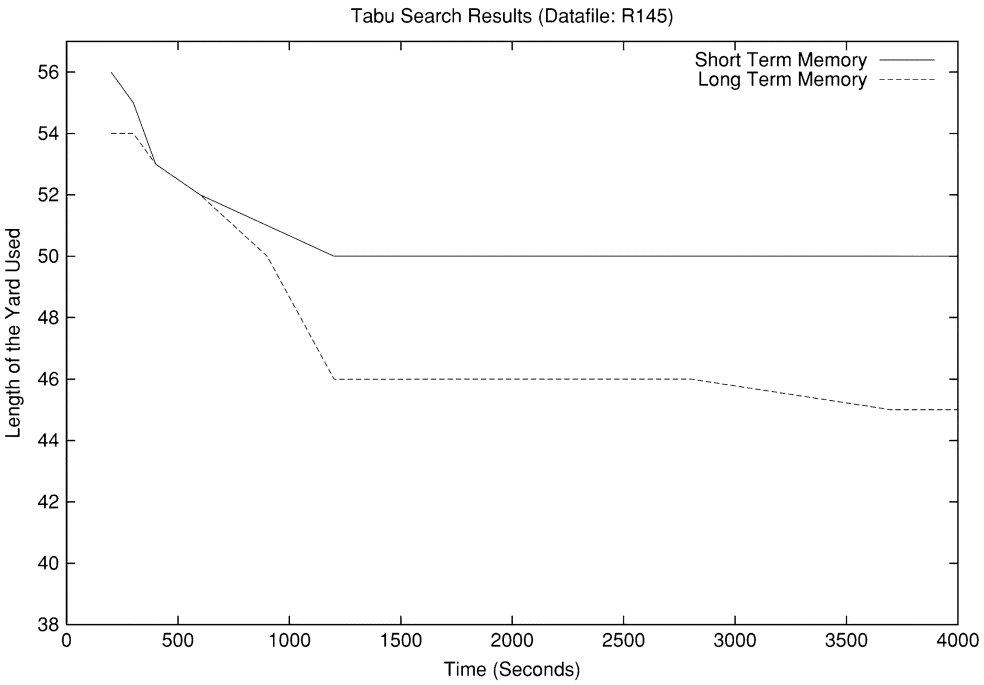


Fig. 14. Results obtained by TS. Data Set: R145.

B. Mutation

Mutation is a well-established genetic operator which alters one or more genes (parts of chromosomes) with a probability equal to the mutation rate. There are several well-known mutation algorithms which work well for different problems. These are:

- inversion: invert a subsequence;
- insertion: select an AL and reinsert it into a random position;
- displacement: select a subsequence and reinsert it into a random position;
- reciprocal exchange: swap two ALs.

In fact, the inversion, displacement and reciprocal exchange mutations are similar to our neighborhood solution and diversification techniques used in TS and SA in this study. Here, we adopted a relatively low mutation rate of 1%.

The population size $P = 1000$ is set for most cases with the evolution process starting with a random population. The population is sorted according to the objective function where the

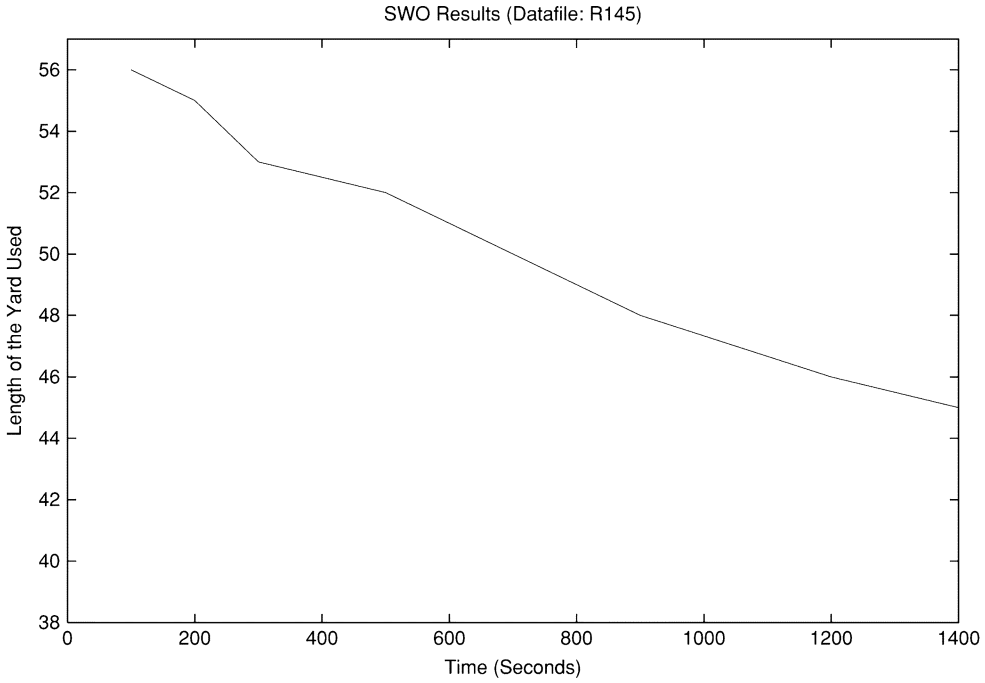


Fig. 15. Results obtained by SWO. Data Set: R145.

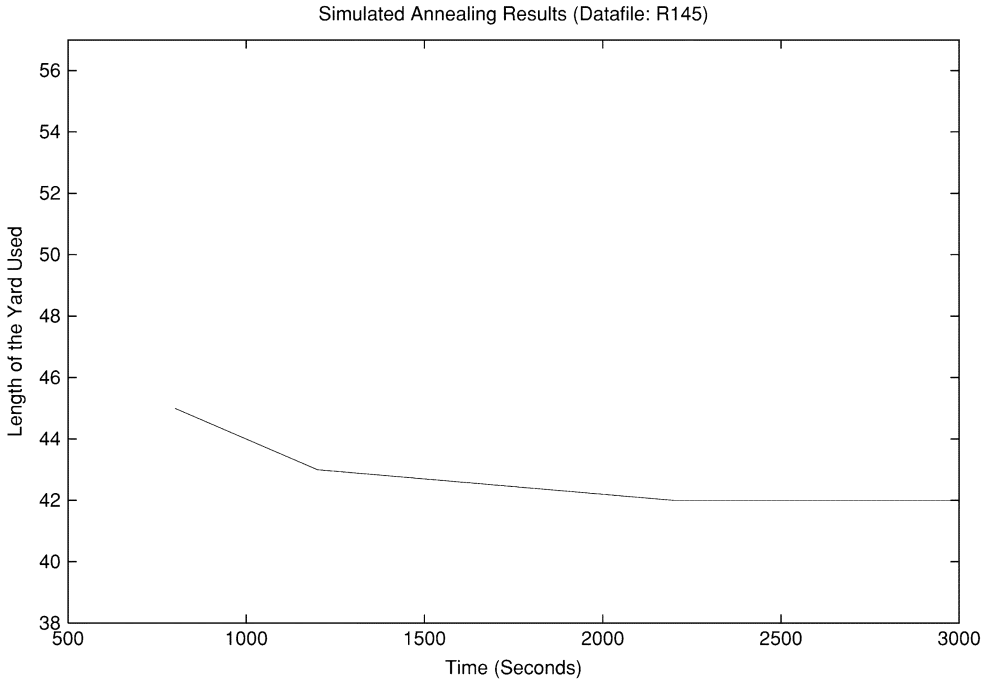


Fig. 16. Results obtained by SA. Data Set: R145.

better the quality, the higher the probability it will be selected for reproduction. At each iteration, a new generation with population size $2P$ is produced and the better half, of size P, survives into the next iteration. This evolution process continues until the stopping criterion is met.

IX. Experimental Results

Real data are not available due to their commercial sensitivity and, even if available, may not be sufficiently challenging for the solution techniques developed. Data sets generated for experiments here overcome this inadequacy since they are designed to be varied and more dense which makes it more difficult for any algorithm to pack. We conducted extensive experiments on randomly generated data.[1] Given a specific yard, space requests are generated to fill up the space. We randomly generated a series of SLSs. These SLSs grow (or remain the same) with time and the rate of this growth is determined by a parameter taken

[1]All data files used in this work can be found at http://www.comp.nus.edu.sg/~fuzh/YAP.

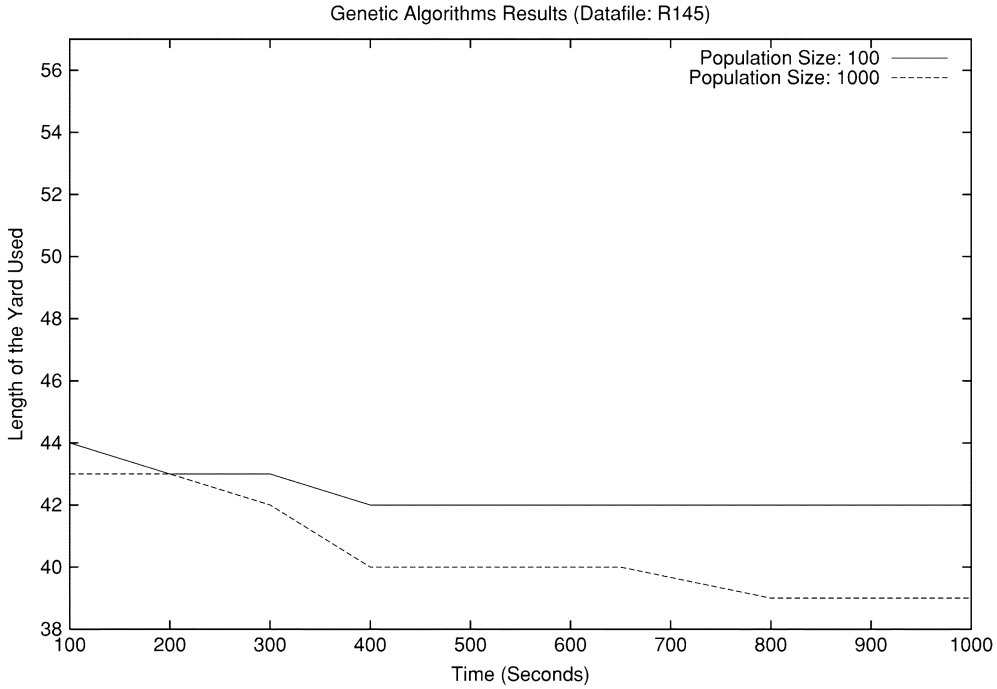


Fig. 17. Results obtained by the GAs. Data Set: R145.

TABLE I
EXPERIMENTAL RESULTS (NAME OF DATA SET SHOWS NUMBER OF SLSs IN FILE. LB: LOWER BOUND, STM: TSSTM, LTM: TSLTM, SWO: SWO, S + T: SWO COMBINED WITH STM, SA: SA, AND GA: GAs)

Data	LB	STM	LTM	SWO	S+T	SA	GA
R126	21	28	26	25	24	25	24
R117	34	39	37	36	34	34	34
R145	39	50	45	45	40	42	39
R178	50	69	69	67	58	66	55
R188	74	105	98	98	82	102	79
R173	77	98	91	94	83	98	79
R250	83	141	119	113	94	117	89
R236	97	139	130	133	107	114	101
R213	164	245	246	246	190	245	187
Average	71	101	96	95	79	94	76

TABLE II
RUNNING TIME (SECONDS) FOR TABLE I

Data	STM	LTM	SWO	S+T	SA	GA
R126	594	3423	1014	2719	2023	302
R117	753	2521	956	2821	1873	442
R145	1215	3693	1342	3375	2233	784
R178	2568	5362	3474	5890	2576	1023
R188	2523	7822	2832	7832	3529	1321
R173	3432	6743	2764	7292	3431	1675
R250	4578	10239	3598	15632	5031	3632
R236	5027	11053	3486	17021	5892	2453
R213	4891	10476	3632	18983	6342	4322
Average	2842	6815	2566	9063	3659	1773

to be linear with time. SLSs are generated in two phases. In the first phase, they are generated randomly. In the second phase, we calculate a simple lower bound for each time slot and additional SLSs' are generated according to these lower bounds. Figs. 10–17 provide the representation graphs for each of the test cases and each contains one connected component—in other

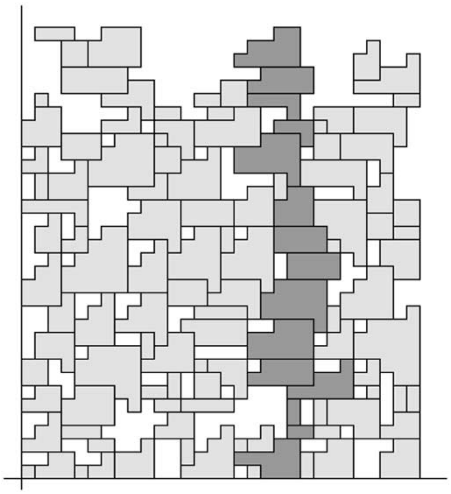


Fig. 18. Physical layout of 117 SLSs (requests). Data Set: R117.

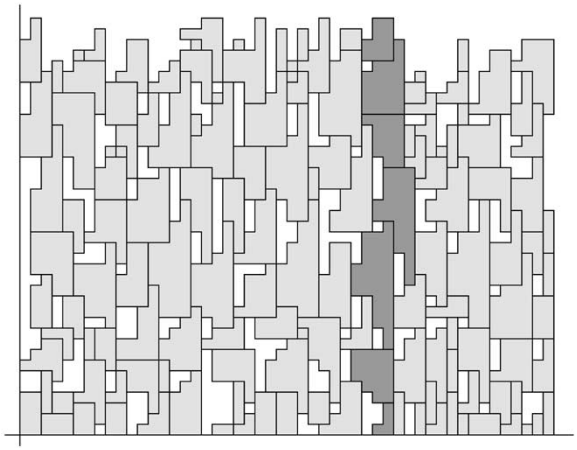


Fig. 19. Physical layout of 145 SLSs (requests). Data Set: R145.

words, the test cases cannot be partitioned into more than one independent subcase. Because the problem involves sliding SLSs and special constraints, we are able to determine only a simple lower bound which is taken to be the sum of the space requirements at each time slot. Data sets used vary according to the size of the yards and contain varying numbers of SLSs.

A. Parameter Tuning

Each of the previously-mentioned heuristic methods require several input parameters, which need to be tuned carefully. Most of the parameter settings we use are given in previous sections according to each of the heuristic approaches. We usually select the most cost-effective cutoff point of the performance curves. Figs. 10–17 are provided so as to visualize certain parameter tuning for the various heuristic algorithms we have implemented and applied to the two data sets R117 and R145.

B. Results and Analysis

Table I gives the experimental results. Here, the method GA outperforms all the other heuristics in all test cases by a considerable margin. TSSTM has the simplest implementation but gives the worst results. TSLTM provides improvement to TSSTM but the level of improvement is not stable. We believe one of the major difficulties with LTM is with the assignment of the relative weights to yard length, residence frequency and transition frequency in the objective function. SWO is relatively easy to implement with comparable results to TSLTM. Experiments show SWO has good *diversification* while TS is good at *intensification*. The combination of the two methods complement each other in SWO + TS which gives the best results and are within 10% of the lower bound. SA is relatively easy to implement giving results comparable to TSLTM. The most successful approach, using a GA, gives the best results, which are within 8% of the simple lower bound for most cases. Average performance results are also given for all algorithms.

Table II shows the running time for each of the tests performed in Table I on a Dual-CPU (Pentium III 800 MHz each) Linux machine. It is clear that GA is the most time cost-effective approach.

An interesting finding from the experiments was that normalization does not help TS and GA significantly; besides slowing these down, it sometimes caused degenerate results. Although normalization should cause the search process to be more focused, its side effects can outweigh its merits. However, normalization does improve the performance of SWO greatly by reducing the search space. When this happens, the search process becomes more stable and focused.

Figs. 18 and 19 provide results obtained by GA for input file R117 and R145.[2] The heavily shaded SLSs contain the region that is the densest.

X. Conclusion

In this work, we studied a basic port yard storage problem that was motivated by logistics management needs at the Port of Singapore. The PYSOP is NP-hard. As a first step to solution, a geometrical representation of PYSOP was transformed into a DAG for efficient manipulation. A normalization procedure was proposed to guarantee a one-to-one relationship between geometric layout and ALs, following which the optimal layout optimization problem is transformed to a search for the optimal AL assignment.

Several heuristic methods including TS, SA, SWO and GAs were then applied with adaptations to suite the PYSOP. Extensive experiments were conducted using data sets that were sufficiently diverse and challenging and results show that the GA approach achieves best results within relatively short times for most cases. A combination of the traditional TS with the recently-developed SWO also gives interesting results, although these come short of the results using the GA approach. These two methods outperformed all other heuristics by a margin of 10%. In all, we found solution techniques which give good solutions to the PYSOP and are easily deployed to operational situations.

[2]The graphical results with 200+ requests are too large to fit in, more graphs can be found at the webpage mentioned in an earlier footnote.

References

[1] F. Sabria and C. Daganzo, "Queuing systems with scheduled arrivals and established service order," *Transport. Res. B*, vol. 23, pp. 159–175, 1989.

[2] E. K. Bish, T.-Y. Leong, C.-L. Li, J. W. C. Ng, and D. Simchi-Levi, "Analysis of a new vehicle scheduling and location problem," *Nav. Rese. Logist.*, vol. 48, pp. 363–385, 2001.

[3] L. Gambardella, A. Rizzoli, and M. Zaffalon, "Simulation and planning of an intermodal container terminal," *Simulation*, vol. 71, pp. 107–116, 1998.

[4] A. Lim, "On the ship berthing problem," *Oper. Res. Lett.*, vol. 22, no. 2–3, pp. 105–110, 1998.

[5] ——, "An effective ship berthing algorithm," in *Proc. 16th Int. Joint Conf. Artificial Intelligence*, vol. 1, 1999, pp. 594–599.

[6] K. S. Goh, Z. Fu, and A. Lim, "A GA based approach for the ship berthing problem," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2000, pp. 465–471.

[7] Z. Fu and A. Lim, "A hybrid method for the ship berthing problem," in *Proc. 6th Conf. Artificial Intelligence and Soft Computing*, 2000, pp. 87–93.

[8] E. Hopper and B. Turton, "A review of the application of meta-heuristic algorithms to 2d strip packing problems," *Artif. Intell. Rev.*, vol. 16, pp. 257–300, 2001.

[9] F. Glover and M. Laguna, *Tabu Search*. Norwell, MA: Kluwer, 1997.

[10] P. L. Hammer, *Tabu Search*, Basel, Switzerland: J. C. Baltzer, 1993.

[11] S. Sait and H. Youssef, *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. New York: IEEE, 1999.

[12] D. Pham and D. Karaboga, *Intelligent Optimization Techniques: Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. New York: Springer, 2000.

[13] B. Hajek, "Cooling schedules for optimal annealing," *Math. Oper. Res.*, vol. 13, pp. 311–329, 1988.

[14] S. Kirpatrick, C. Gelatt Jr., and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[15] R. H. J. M. Otten and L. P. P. P. V. Ginneken, *The Annealing Algorithm*. Boston, MA: Kluwer, 1989.

[16] D. E. Joslin and D. P. Clements, "Squeaky wheel optimization," *J. Artif. Intell. Res.*, vol. 10, pp. 353–373, 1999.

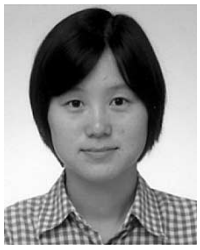
[17] D. E. Joslin and D. P. Clements, "Squeaky wheel optimization," in *Proc. 15th Nat. Conf. Artificial Intelligence (AAAI'98)*, Madison, WI, 1998, pp. 340–346.

[18] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz, and M. Savelsbergh, "Heuristic optimization: A hybrid AI/OR approach," in *Proc. Workshop Industrial Constraint-Directed Scheduling*, 1997.

[19] D. Draper, A. Jonsson, D. Clements, and D. Joslin, "Cyclic scheduling," in *Proc. 16th Int. Joint Conf. Artificial Intelligence*, vol. 2, 1999, pp. 1016–1021.

[20] J. H. Holland, *Adaptation in Natural Artificial Systems*. Ann Arbor, MI: Univ. of Michigan Press, 1975.

[21] Z. Michalewicz, *Genetic Algorithms + Data Structure = Evolution Programs*. New York: Springer-Verlag, 1996.
[22] D. Goldberg and R. Lingle, "Alleles Loci and the traveling salesman problem," in *Proc. 2nd Int. Conf. Genetic Algorithms*, 1985, pp. 154–159.
[23] I. Oliver, D. Smith, and J. Holland, "A study of permutation crossover operators on the TSP," in *Proc. 2nd Int. Conf. Genetic Algorithms Applications*, Cambridge, MA, 1987, pp. 224–230.



Ping Chen received the B.Sc. degree (first-class honors) in computer science from the National University of Singapore, Sinagpore, in 2001. She is working toward the doctoral degree at the University of Maryland, College Park.

Zhaohui Fu (S'99) received the B.S. degree (with first class honors) and M.S. degree in computer science from the National University of Singapore, Singapore, in 2000 and 2002, respectively. He is currently working toward the Ph.D. degree in electrical engineering at Princeton University, Princeton, NJ.

His main research interests are in Boolean satisfiability, combinatorial optimization and design automation. He is working with the SAT research group on the development of efficient SAT and quantified Boolean formula (QBF) solvers.

Andrew Lim (SM'00) received the Ph.D. degree from the University of Minnesota, Minneapolis, in 1992.

From 1992 to 1997, he was a System Developer, Project Leader, and Consultant to many large private and government organizations in Singapore. From 1997 to 2002, he was an Associate Professor of Computer Science at the National University of Singapore. Currently, he is an Associate Professor in the Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology. His interests include algorithms, software components, and framework and architecture for global supply chain management. He has published more than 150 papers in premier international conferences and journals. From 1999 to 2001, he was the coach and team leader of the Singapore International Olympiad in Informatics (IOI) team. His leadership has culminated in Singapore being ranked first out of 74 countries in IOI–2001. From 1998 to 2003, he was also the coach of the Association for Computing Machinery Programming Team which is consistently one of the top teams in Asia.

Brian Rodrigues received the Ph.D. degree in mathematics from the University of California, Santa Barbara, in 1987.

He worked in academia in the U.S. for a few years. He was the Deputy Head of the Operations Analysis Department and a Senior Research Scientist for the Ministry of Defense, Singapore, from 1992 to 1995. He was subsequently employed by the National University of Singapore and after a brief stint, he moved to the Singapore Management University where he is currently an Associate Professor. His interests and expertise include operations research/ management science, artificial intelligence and quality management. He has published a number of papers in international refereed journals and conferences and serves as a Technical Consultant.